Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	3931	((713/155,156,164,170,176) or (380/30,259,282,279,285)).CCLS.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/09 10:25
L2	223	L1 and ticket	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/09 10:25
L3	140	L1 and kerberos	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/09 10:25
L4	6	(("5,809,144") or ("5,724,425") or ("5,923,756")).PN.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/06/09 10:25

Network Working Group   J. Kohl
Request for Comments: 1510   Digital Equipment Corporation
                                                              C. Neuman
                                                                    ISI
                                                         September 1993

The Kerberos Network Authentication Service (V5)

Status of this Memo

Abstract

This document gives an overview and specification of Version 5 of the protocol for the Kerberos network authentication system. Version 4, described elsewhere [1,2], is presently in production use at MIT's Project Athena, and at other Internet sites.

Overview

Project Athena, Athena, Athena MUSE, Discuss, Hesiod, Kerberos, Moira, and Zephyr are trademarks of the Massachusetts Institute of Technology (MIT). No commercial use of these trademarks may be made without prior written permission of MIT.

This RFC describes the concepts and model upon which the Kerberos network authentication system is based. It also specifies Version 5 of the Kerberos protocol.

The motivations, goals, assumptions, and rationale behind most design decisions are treated cursorily; for Version 4 they are fully described in the Kerberos portion of the Athena Technical Plan [1]. The protocols are under review, and are not being submitted for consideration as an Internet standard at this time. Comments are encouraged. Requests for addition to an electronic mailing list for discussion of Kerberos, kerberos@MIT.EDU, may be addressed to kerberos-request@MIT.EDU. This mailing list is gatewayed onto the Usenet as the group comp.protocols.kerberos. Requests for further information, including documents and code availability, may be sent to info-kerberos@MIT.EDU.

Background

The Kerberos model is based in part on Needham and Schroeder's trusted third-party authentication protocol [3] and on modifications suggested by Denning and Sacco [4]. The original design and implementation of Kerberos Versions 1 through 4 was the work of two former Project Athena staff members, Steve Miller of Digital Equipment Corporation and Clifford Neuman (now at the Information Sciences Institute of the University of Southern California), along with Jerome Saltzer, Technical Director of Project Athena, and Jeffrey Schiller, MIT Campus Network Manager. Many other members of Project Athena have also contributed to the work on Kerberos. Version 4 is publicly available, and has seen wide use across the Internet.

Version 5 (described in this document) has evolved from Version 4 based on new requirements and desires for features not available in Version 4. Details on the differences between Kerberos Versions 4 and 5 can be found in [5].

Table of Contents

1. Introduction

Kerberos provides a means of verifying the identities of principals, (e.g., a workstation user or a network server) on an open (unprotected) network. This is accomplished without relying on authentication by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will. (Note, however, that many applications use Kerberos' functions only upon the initiation of a stream-based network connection, and assume the absence of any "hijackers" who might subvert such a connection. Such use implicitly trusts the host addresses involved.) Kerberos performs authentication under these conditions as a trusted third-party authentication service by using conventional cryptography, i.e., shared secret key. (shared secret key - Secret and private are often used interchangeably in the literature. In our usage, it takes two (or more) to share a secret, thus a shared DES key is a secret key. Something is only private when no one but its owner knows it. Thus, in public key cryptosystems, one has a public and a private key.)

The authentication process proceeds as follows: A client sends a request to the authentication server (AS) requesting "credentials" for a given server. The AS responds with these credentials, encrypted in the client's key. The credentials consist of 1) a "ticket" for the server and 2) a temporary encryption key (often called a "session key"). The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the server. The session key (now shared by the client and server) is used to authenticate the client, and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

The implementation consists of one or more authentication servers running on physically secure hosts. The authentication servers maintain a database of principals (i.e., users and servers) and their secret keys. Code libraries provide encryption and implement the Kerberos protocol. In order to add authentication to its

transactions, a typical network application adds one or two calls to the Kerberos library, which results in the transmission of the necessary messages to achieve authentication.

The Kerberos protocol consists of several sub-protocols (or exchanges). There are two methods by which a client can ask a Kerberos server for credentials. In the first approach, the client sends a cleartext request for a ticket for the desired server to the AS. The reply is sent encrypted in the client's secret key. Usually this request is for a ticket-granting ticket (TGT) which can later be used with the ticket-granting server (TGS). In the second method, the client sends a request to the TGS. The client sends the TGT to the TGS in the same manner as if it were contacting any other application server which requires Kerberos credentials. The reply is encrypted in the session key from the TGT.

The expiration time of the ticket will be set to the minimum of the following:

+The expiration time (endtime) requested in the KRB_AS_REQ message.

+The ticket's start time plus the maximum allowable lifetime associated with the client principal (the authentication server's database includes a maximum ticket lifetime field in each principal's record; see section 4).

+The ticket's start time plus the maximum allowable lifetime associated with the server principal.

+The ticket's start time plus the maximum lifetime set by the policy of the local realm.

If the requested expiration time minus the start time (as determined above) is less than a site-determined minimum lifetime, an error message with code KDC_ERR_NEVER_VALID is returned. If the requested expiration time for the ticket exceeds what was determined as above, and if the "RENEWABLE-OK" option was requested, then the "RENEWABLE" flag is set in the new ticket, and the renew-till value is set as if the "RENEWABLE" option were requested (the field and option names are described fully in section 5.4.1). If the RENEWABLE option has been requested or if the RENEWABLE-OK option has been set and a renewable ticket is to be issued, then the renew-till field is set to the minimum of:

+Its requested value.

+The start time of the ticket plus the minimum of the two maximum renewable lifetimes associated with the principals' database entries.

+The start time of the ticket plus the maximum renewable lifetime set by the policy of the local realm.

The flags field of the new ticket will have the following options set if they have been requested and if the policy of the local realm allows: FORWARDABLE, MAY-POSTDATE, POSTDATED, PROXIABLE, RENEWABLE. If the new ticket is postdated (the start time is in the future), its INVALID flag will also be set.

If all of the above succeed, the server formats a KRB_AS_REP message (see section 5.4.2), copying the addresses in the request into the caddr of the response, placing any required pre-authentication data into the padata of the response, and encrypts the ciphertext part in the client's key using the requested encryption method, and sends it to the client. See section A.2 for pseudocode.

3.1.4. Generation of KRB_ERROR message

Several errors can occur, and the Authentication Server responds by returning an error message, KRB_ERROR, to the client, with the error-code and e-text fields set to appropriate values. The error message contents and details are described in Section 5.9.1.

3.1.5. Receipt of KRB_AS_REP message

If the reply message type is KRB_AS_REP, then the client verifies that the cname and crealm fields in the cleartext portion of the reply match what it requested. If any padata fields are present, they may be used to derive the proper secret key to decrypt the message. The client decrypts the encrypted part of the response using its secret key, verifies that the nonce in the encrypted part matches the nonce it supplied in its request (to detect replays). It also verifies that the sname and srealm in the response match those in the request, and that the host address field is also correct. It then stores the ticket, session key, start and expiration times, and other information for later use. The key-expiration field from the encrypted part of the response may be checked to notify the user of impending key expiration (the client program could then suggest remedial action, such as a password change). See section A.3 for pseudocode.

Proper decryption of the KRB_AS_REP message is not sufficient to

verify the identity of the user; the user and an attacker could cooperate to generate a KRB_AS_REP format message which decrypts properly but is not from the proper KDC. If the host wishes to verify the identity of the user, it must require the user to present application credentials which can be verified using a securely-stored secret key. If those credentials can be verified, then the identity of the user can be assured.

3.1.6. Receipt of KRB_ERROR message

If the reply message type is KRB_ERROR, then the client interprets it as an error and performs whatever application-specific tasks are necessary to recover.

3.2. The Client/Server Authentication Exchange

Summary

Message direction	Message type	Section
Client to Application server	KRB_AP_REQ	5.5.1
[optional] Application server to client	KRB_AP_REP or	5.5.2
	KRB_ERROR	5.9.1

The client/server authentication (CS) exchange is used by network applications to authenticate the client to the server and vice versa. The client must have already acquired credentials for the server using the AS or TGS exchange.

3.2.1. The KRB_AP_REQ message

The KRB_AP_REQ contains authentication information which should be part of the first message in an authenticated transaction. It contains a ticket, an authenticator, and some additional bookkeeping information (see section 5.5.1 for the exact format). The ticket by itself is insufficient to authenticate a client, since tickets are passed across the network in cleartext(Tickets contain both an encrypted and unencrypted portion, so cleartext here refers to the entire unit, which can be copied from one message and replayed in another without any cryptographic skill.), so the authenticator is used to prevent invalid replay of tickets by proving to the server that the client knows the session key of the ticket and thus is entitled to use it. The KRB_AP_REQ message is referred to elsewhere as the "authentication header."

3.2.2. Generation of a KRB_AP_REQ message

When a client wishes to initiate authentication to a server, it obtains (either through a credentials cache, the AS exchange, or the

TGS exchange) a ticket and session key for the desired service. The padata field, and including the same fields as used in the KRB_AS_REQ message along with several optional fields: the enc-authorization-data field for application server use and additional tickets required by some options.

In preparing the authentication header, the client can select a sub-session key under which the response from the Kerberos server will be encrypted (If the client selects a sub-session key, care must be taken to ensure the randomness of the selected subsession key. One approach would be to generate a random number and XOR it with the session key from the ticket-granting ticket.). If the sub-session key is not specified, the session key from the ticket-granting ticket will be used. If the enc-authorization-data is present, it must be encrypted in the sub-session key, if present, from the authenticator portion of the authentication header, or if not present in the session key from the ticket-granting ticket.

Once prepared, the message is sent to a Kerberos server for the destination realm. See section A.5 for pseudocode.

3.3.2. Receipt of KRB_TGS_REQ message

The KRB_TGS_REQ message is processed in a manner similar to the KRB_AS_REQ message, but there are many additional checks to be performed. First, the Kerberos server must determine which server the accompanying ticket is for and it must select the appropriate key to decrypt it. For a normal KRB_TGS_REQ message, it will be for the

ticket granting service, and the TGS's key will be used. If the TGT was issued by another realm, then the appropriate inter-realm key must be used. If the accompanying ticket is not a ticket granting ticket for the current realm, but is for an application server in the current realm, the RENEW, VALIDATE, or PROXY options are specified in the request, and the server for which a ticket is requested is the server named in the accompanying ticket, then the KDC will decrypt the ticket in the authentication header using the key of the server for which it was issued. If no ticket can be found in the padata field, the KDC_ERR_PADATA_TYPE_NOSUPP error is returned.

Once the accompanying ticket has been decrypted, the user-supplied checksum in the Authenticator must be verified against the contents of the request, and the message rejected if the checksums do not match (with an error code of KRB_AP_ERR_MODIFIED) or if the checksum is not keyed or not collision-proof (with an error code of KRB_AP_ERR_INAPP_CKSUM). If the checksum type is not supported, the KDC_ERR_SUMTYPE_NOSUPP error is returned. If the authorization-data are present, they are decrypted using the sub-session key from the Authenticator.

If any of the decryptions indicate failed integrity checks, the KRB_AP_ERR_BAD_INTEGRITY error is returned.

3.3.3. Generation of KRB_TGS_REP message

The KRB_TGS_REP message shares its format with the KRB_AS_REP (KRB_KDC_REP), but with its type field set to KRB_TGS_REP. The detailed specification is in section 5.4.2.

The response will include a ticket for the requested server. The Kerberos database is queried to retrieve the record for the requested server (including the key with which the ticket will be encrypted). If the request is for a ticket granting ticket for a remote realm, and if no key is shared with the requested realm, then the Kerberos server will select the realm "closest" to the requested realm with which it does share a key, and use that realm instead. This is the only case where the response from the KDC will be for a different server than that requested by the client.

By default, the address field, the client's name and realm, the list of transited realms, the time of initial authentication, the expiration time, and the authorization data of the newly-issued ticket will be copied from the ticket-granting ticket (TGT) or renewable ticket. If the transited field needs to be updated, but the transited type is not supported, the KDC_ERR_TRTYPE_NOSUPP error is returned.

If the request specifies an endtime, then the endtime of the new ticket is set to the minimum of (a) that request, (b) the endtime from the TGT, and (c) the starttime of the TGT plus the minimum of the maximum life for the application server and the maximum life for the local realm (the maximum life for the requesting principal was already applied when the TGT was issued). If the new ticket is to be a renewal, then the endtime above is replaced by the minimum of (a) the value of the renew_till field of the ticket and (b) the starttime for the new ticket plus the life (endtimestarttime) of the old ticket.

If the FORWARDED option has been requested, then the resulting ticket will contain the addresses specified by the client. This option will only be honored if the FORWARDABLE flag is set in the TGT. The PROXY option is similar; the resulting ticket will contain the addresses specified by the client. It will be honored only if the PROXIABLE flag in the TGT is set. The PROXY option will not be honored on requests for additional ticket-granting tickets.

If the requested start time is absent or indicates a time in the past, then the start time of the ticket is set to the authentication server's current time. If it indicates a time in the future, but the POSTDATED option has not been specified or the MAY-POSTDATE flag is not set in the TGT, then the error KDC_ERR_CANNOT_POSTDATE is returned. Otherwise, if the ticket-granting ticket has the MAYPOSTDATE flag set, then the resulting ticket will be postdated and the requested starttime is checked against the policy of the local realm. If acceptable, the ticket's start time is set as requested, and the INVALID flag is set. The postdated ticket must be validated before use by presenting it to the KDC after the starttime has been reached. However, in no case may the starttime, endtime, or renew-till time of a newly-issued postdated ticket extend beyond the renew-till time of the ticket-granting ticket.

If the ENC-TKT-IN-SKEY option has been specified and an additional ticket has been included in the request, the KDC will decrypt the additional ticket using the key for the server to which the additional ticket was issued and verify that it is a ticket-granting ticket. If the name of the requested server is missing from the request, the name of the client in the additional ticket will be used. Otherwise the name of the requested server will be compared to the name of the client in the additional ticket and if different, the request will be rejected. If the request succeeds, the session key from the additional ticket will be used to encrypt the new ticket that is issued instead of using the key of the server for which the new ticket will be used (This allows easy implementation of user-to-user authentication [6], which uses ticket-granting ticket session keys in lieu of secret server keys in situations where such secret

keys could be easily compromised.).

If the name of the server in the ticket that is presented to the KDC as part of the authentication header is not that of the ticket-granting server itself, and the server is registered in the realm of the KDC, If the RENEW option is requested, then the KDC will verify that the RENEWABLE flag is set in the ticket and that the renew_till time is still in the future. If the VALIDATE option is rqeuested, the KDC will check that the starttime has passed and the INVALID flag is set. If the PROXY option is requested, then the KDC will check that the PROXIABLE flag is set in the ticket. If the tests succeed, the KDC will issue the appropriate new ticket.

Whenever a request is made to the ticket-granting server, the presented ticket(s) is(are) checked against a hot-list of tickets which have been canceled. This hot-list might be implemented by storing a range of issue dates for "suspect tickets"; if a presented ticket had an authtime in that range, it would be rejected. In this way, a stolen ticket-granting ticket or renewable ticket cannot be used to gain additional tickets (renewals or otherwise) once the theft has been reported. Any normal ticket obtained before it was reported stolen will still be valid (because they require no interaction with the KDC), but only until their normal expiration time.

The ciphertext part of the response in the KRB_TGS_REP message is encrypted in the sub-session key from the Authenticator, if present, or the session key key from the ticket-granting ticket. It is not encrypted using the client's secret key. Furthermore, the client's key's expiration date and the key version number fields are left out since these values are stored along with the client's database record, and that record is not needed to satisfy a request based on a ticket-granting ticket. See section A.6 for pseudocode.

3.3.3.1. Encoding the transited field

If the identity of the server in the TGT that is presented to the KDC as part of the authentication header is that of the ticket-granting service, but the TGT was issued from another realm, the KDC will look up the inter-realm key shared with that realm and use that key to decrypt the ticket. If the ticket is valid, then the KDC will honor the request, subject to the constraints outlined above in the section describing the AS exchange. The realm part of the client's identity will be taken from the ticket-granting ticket. The name of the realm that issued the ticket-granting ticket will be added to the transited field of the ticket to be issued. This is accomplished by reading the transited field from the ticket-granting ticket (which is treated as an unordered set of realm names), adding the new realm to the set,

then constructing and writing out its encoded (shorthand) form (this may involve a rearrangement of the existing encoding).

Note that the ticket-granting service does not add the name of its own realm. Instead, its responsibility is to add the name of the previous realm. This prevents a malicious Kerberos server from intentionally leaving out its own name (it could, however, omit other realms' names).

The names of neither the local realm nor the principal's realm are to be included in the transited field. They appear elsewhere in the ticket and both are known to have taken part in authenticating the principal. Since the endpoints are not included, both local and single-hop inter-realm authentication result in a transited field that is empty.

Because the name of each realm transited is added to this field, it might potentially be very long. To decrease the length of this field, its contents are encoded. The initially supported encoding is optimized for the normal case of inter-realm communication: a hierarchical arrangement of realms using either domain or X.500 style realm names. This encoding (called DOMAIN-X500-COMPRESS) is now